REMARKS/ARGUMENTS

Claim Rejections - 35 USC § 112

Claims 6, 7 and 14 were rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

As for claims 6 and 14, the Examiner was unclear compared to what entity in the claimed invention does the data memory have a higher priority during hit/miss determination.

The Examiner correctly interpreted the meaning of Claims 6 and 14 in the Office Action of June 25, 2006, pages 5 - 6, item 7. The claims are based on the specification paragraph [0049]. Therefore, since the Examiner understood the meaning, Claims 6 and 14 are believed to satisfy 35 U.S.C. 112, second paragraph and his query.

The Examiner stated there is insufficient antecedent basis in Claim 7 for "the 2-way set associative cache".

Claim 7 is amended and there is now antecedent basis for the multi-way set associative cache.

Claim Rejections - 35 USC § 102

Claims 1, 3,7, 9, 11, 15-20 and 22 were rejected under U.S.C. 102(b) as being anticipated by Tremblay et al. [6,125,439].

Applicants traverse the rejection of Claim 1. The Examiner stated " a cache subsystem comprising a multi-way set associative cache (Fig 1, 165; Col 20, Lines 54-59) and a data memory that holds a contiguous block of memory (Col 17, Lines 11-13; Fig 4A, Stack 400) defined by an address stored in a register (Col 8, Lines 10-20), wherein local variables are stored in said data memory (Col 8, Lines 50-60; Col 9, Lines 55-60)." As the Examiner pointed out, in Tremblay, the local variables are stored on the stack cache 400 (155 of Tremblay Fig. 1) and not in the data cache 165 of Tremblay Fig. 1 which is the "data memory" unit recited in Claim 1 (126 of 122 in Fig. 3 of the present application). To clarify this distinction, Claim 1 is amended to identify which cache is being used for local variable storage. Claim 1 now recites "a data cache subsystem"; the word "data" is supported by the first line of paragraph [0040] stating "data storage 122" of Fig. 3 and the paragraph then identifies element 126 of Fig. 3 to be the "contiguous block of memory" that stores "local variables" (4^{th} line in [0040]). Further, does not use his data cache 165/160 to store local variables because he stores local variables in a dedicated local variable storage area (Col 5 L6-16, Col 8 L58-59) that is located on stack 400, the stack cache, Fig. 4A, element 155.

Applicants note further, Tremblay not only stores his local variables in a stack cache 155 of 150 (Fig. 1) rather than the "data memory" recited in Claim 1, but also that Tremblay does not anticipate Claim 1's "multi-way set associative cache". Tremblay's Fig. 1, element 150, does not have an associative cache. Instead, Tremblay states his elements 160, 145, 146 and 147 have associative caches. (See Col 20 - L54, Col 22 - L21, Col 23 - L66, Col 23 – L47). Since Tremblay meticulously describes these other units as having associative cache, it is clear that since Tremblay does not mention stack 150 as having associative cache, then 150 indeed does not have associative cache. A copy of a memo defining associative cache from www.pcguide.com is provided for the Examiner, attached to the end of this amendment.

Because of the above reasons, Applicants believe that Claim 1 is allowable

without any further amendments. However, to speedily advance Claim 1 to an allowance, the modifier "data" is now included.

Claims 2 – 8, being dependent on Claim 1 which should be allowable, should also be allowable for the at least reasons stated for Claim 1. Furthermore, Claims 2, 4, 5 and 8 have elements which the Examiner already found to be novel in the Office Action of April 25, 2006.

Claims 9, 17 and 20 are cancelled and the reasons for rejection no longer apply. Claims 11, 15, 16, 18-19, 22 now depend on base claims that are amended in a form allowed by the Examiner. Therefore the reasons for rejection for Claims 11, 15, 16, 18-19, 22, no longer apply.

Claim Rejections - 35 USC § 103

Claims 6 and 14 were rejected under 35 U.S.C. 103(a) as being unpatentable over Tremblay et al.

Claim 6, dependent on Claim 1, is believed allowable for the at least same reasons cited above for Claim 1. Claim 14 now depends on Claim 13, amended in a form allowable by the Examiner. Therefore the reason for rejection for Claim 14 no longer applies.

Allowable Subject Matter

Applicants thank the Examiner for the allowed claims.
Claims 2, 4, 5, 8, 10, 12, 13, 21 and 23 were objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent

form including all of the limitations of the base claim and any intervening claims. Claims 10, 12, 13, 21 and 23 are thus amended.

Respectful request is made for a continued examination (RCE), reconsideration of Claim 1, and issuing a Notice of Allowance.

Respectfully submitted,

/Dolly Y. Wu/
Dolly Y. Wu
Reg. No. 59,192
Texas Instruments Incorporated
PO Box 655474, M/S 3999
Dallas, Texas 75265
972.917.4144

Cache Mapping and Associativity

A very important factor in determining the effectiveness of the level 2 cache relates to how the cache is mapped to the system memory. What this means in brief is that there are many different ways to allocate the storage in our cache to the memory addresses it serves. Let's take as an example a system with 512 KB of L2 cache and 64 MB of main memory. The burning question is: how do we decide how to divvy up the 16,384 address lines in our cache amongst the "huge" 64 MB of memory?

There are three different ways that this mapping can generally be done. The choice of mapping technique is so critical to the design that the cache is often named after this choice:

- **Direct Mapped Cache:** The simplest way to allocate the cache to the system memory is to determine how many cache lines there are (16,384 in our example) and just chop the system memory into the same number of chunks. Then each chunk gets the use of one cache line. This is called *direct mapping*. So if we have 64 MB of main memory addresses, each cache line would be shared by 4,096 memory addresses (64 M divided by 16 K).
- **Fully Associative Cache:** Instead of hard-allocating cache lines to particular memory locations, it is possible to design the cache so that any line can store the contents of any memory location. This is called *fully associative mapping*.
- **N-Way Set Associative Cache:** "N" here is a number, typically 2, 4, 8 etc. This is a compromise between the direct mapped and fully associative designs. In this case the cache is broken into sets where each set contains "N" cache lines, let's say 4. Then, each memory address is assigned a set, and can be cached in any one of those 4

locations within the set that it is assigned to. In other words, *within each set* the cache is associative, and thus the name.
This design means that there are "N" possible places that a given memory location may be in the cache. The tradeoff is that there are "N" times as many memory locations competing for the same "N" lines in the set. Let's suppose in our example that we are using a 4-way set associative cache. So instead of a single block of 16,384 lines, we have 4,096 sets with 4 lines in each. Each of these sets is shared by 16,384 memory addresses (64 M divided by 4 K) instead of 4,096 addresses as in the case of the direct mapped cache. So there is more to share (4 lines instead of 1) but more addresses sharing it (16,384 instead of 4,096).

Conceptually, the direct mapped and fully associative caches are just "special cases" of the N-way set associative cache. You can set "N" to 1 to make a "1-way" set associative cache. If you do this, then there is only one line per set, which is the same as a direct mapped cache because each memory address is back to pointing to only one possible cache location. On the other hand, suppose you make "N" really large; say, you set "N" to be equal to the number of lines in the cache (16,384 in our example). If you do this, then you only have one set, containing all of the cache lines, and every memory location points to that huge set. This means that any memory address can be in any line, and you are back to a fully associative cache.

Next: Comparison of Cache Mapping Techniques

Home - Search - Topics - Up